EXHIBIT 59

Search this site
- Documentation
- Getting started
- Concepts
  - Overview
  - Cluster Architecture
    - Nodes
    - Communication between Nodes and the Control Plane
    - Controllers
    - Leases
    - Cloud Controller Manager
    - About cgroup v2
    - Kubernetes Self-Healing
    - Container Runtime Interface (CRI)
    - Garbage Collection
    - Mixed Version Proxy
  - Containers
  - Workloads
  - Services, Load Balancing, and
Search this site

Nodes

Kubernetes runs your workload by placing containers into Pods to run on *Nodes*. A node may be a virtual or physical machine, depending on the cluster. Each node is managed by the control plane and contains the services necessary to run Pods.

Typically you have several nodes in a cluster; in a learning or resource-limited environment, you might have only one node.

The components on a node include the kubelet, a container runtime, and the kube-proxy.

Management

There are two main ways to have Nodes added to the API server:

1. The kubelet on a node self-registers to the control plane
2. You (or another human user) manually add a Node object

After you create a Node object, or the kubelet on a node self-registers, the control plane checks whether the new Node object is valid. For example, if you try to create a Node from the following JSON manifest:

```
{
  "kind": "Node",
  "apiVersion": "v1",
  "metadata": {
    "name": "10.240.79.157",
    "labels": {
      "name": "my-first-k8s-node"
    }
  }
```

Node API reference
Edit this page
Create child page
Create an issue
Print entire section
Management
Node name uniqueness
Self-registration of Nodes
Manual Node administration
Node status
Node heartbeats
Node controller
Rate limits on eviction
Resource capacity tracking
Node topology
Swap memory management
What's next

URL
https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp
Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

2. You (or another human user) manually add a Node object

After you create a Node object, or the kubelet on a node self-registers, the control plane checks whether the new Node object is valid. For example, if you try to create a Node from the following JSON manifest:

```
{
  "kind": "Node",
  "apiVersion": "v1",
  "metadata": {
    "name": "10.240.79.157",
    "labels": {
      "name": "my-first-k8s-node"
    }
  }
}
```

Kubernetes creates a Node object internally (the representation). Kubernetes checks that a kubelet has registered to the API server that matches the `metadata.name` field of the Node. If the node is healthy (i.e. all necessary services are running), then it is eligible to run a Pod. Otherwise, that node is ignored for any cluster activity until it becomes healthy.

> **Note:**
> Kubernetes keeps the object for the invalid Node and continues checking to see whether it becomes healthy.
>
> You, or a controller, must explicitly delete the Node object to stop that health checking.

The name of a Node object must be a valid DNS subdomain name.

Node name uniqueness

The name identifies a Node. Two Nodes cannot have the same name at the same time. Kubernetes also assumes that a resource with the same name is the same object. In case of a Node, it is implicitly assumed that an instance using the same name will have the same state (e.g. network settings, root disk contents) and attributes like node labels. This may lead to inconsistencies if an instance was modified without changing

URL
https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp
Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

Node name uniqueness

The name identifies a Node. Two Nodes cannot have the same name at the same time. Kubernetes also assumes that a resource with the same name is the same object. In case of a Node, it is implicitly assumed that an instance using the same name will have the same state (e.g. network settings, root disk contents) and attributes like node labels. This may lead to inconsistencies if an instance was modified without changing its name. If the Node needs to be replaced or updated significantly, the existing Node object needs to be removed from API server first and re-added after the update.

Self-registration of Nodes

When the kubelet flag `--register-node` is true (the default), the kubelet will attempt to register itself with the API server. This is the preferred pattern, used by most distros.

For self-registration, the kubelet is started with the following options:

- `--kubeconfig` - Path to credentials to authenticate itself to the API server.
- `--cloud-provider` - How to talk to a cloud provider to read metadata about itself.
- `--register-node` - Automatically register with the API server.
- `--register-with-taints` - Register the node with the given list of taints (comma separated `<key>=<value>:<effect>`).

  No-op if `register-node` is false.
- `--node-ip` - Optional comma-separated list of the IP addresses for the node. You can only specify a single address for each address family. For example, in a single-stack IPv4 cluster, you set this value to be the IPv4 address that the kubelet should use for the node. See configure IPv4/IPv6 dual stack for details of running a dual-stack cluster.

  If you don't provide this argument, the kubelet uses the node's default IPv4 address, if any; if the node has no IPv4 addresses then the kubelet uses the node's default IPv6 address.
- `--node-labels` - Labels to add when registering the node in the cluster (see label restrictions enforced by the NodeRestriction admission plugin).
- `--node-status-update-frequency` - Specifies how often kubelet posts its node status to the API server.

URL

https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp

Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

node's default IPv6 address.

- `--node-labels` - Labels to add when registering the node in the cluster (see label restrictions enforced by the NodeRestriction admission plugin).
- `--node-status-update-frequency` - Specifies how often kubelet posts its node status to the API server.

When the Node authorization mode and NodeRestriction admission plugin are enabled, kubelets are only authorized to create/modify their own Node resource.

> **Note:**
> As mentioned in the Node name uniqueness section, when Node configuration needs to be updated, it is a good practice to re-register the node with the API server. For example, if the kubelet is being restarted with a new set of `--node-labels`, but the same Node name is used, the change will not take effect, as labels are only set (or modified) upon Node registration with the API server.
>
> Pods already scheduled on the Node may misbehave or cause issues if the Node configuration will be changed on kubelet restart. For example, already running Pod may be tainted against the new labels assigned to the Node, while other Pods, that are incompatible with that Pod will be scheduled based on this new label. Node re-registration ensures all Pods will be drained and properly re-scheduled.

Manual Node administration

You can create and modify Node objects using kubectl.

When you want to create Node objects manually, set the kubelet flag `--register-node=false`.

You can modify Node objects regardless of the setting of `--register-node`. For example, you can set labels on an existing Node or mark it unschedulable.

You can set optional node role(s) for nodes by adding one or more `node-role.kubernetes.io/<role>: <role>` labels to the node where characters of `<role>` are limited by the syntax rules for labels.

Kubernetes ignores the label value for node roles; by convention, you can set it to the same string you used for the node role in the label key.

URL
https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp
Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

You can set optional node role(s) for nodes by adding one or more `node-role.kubernetes.io/<role>: <role>` labels to the node where characters of `<role>` are limited by the syntax rules for labels.

Kubernetes ignores the label value for node roles; by convention, you can set it to the same string you used for the node role in the label key.

You can use labels on Nodes in conjunction with node selectors on Pods to control scheduling. For example, you can constrain a Pod to only be eligible to run on a subset of the available nodes.

Marking a node as unschedulable prevents the scheduler from placing new pods onto that Node but does not affect existing Pods on the Node. This is useful as a preparatory step before a node reboot or other maintenance.

To mark a Node unschedulable, run:

```
kubectl cordon $NODENAME
```

See Safely Drain a Node for more details.

> **Note:**
> Pods that are part of a DaemonSet tolerate being run on an unschedulable Node. DaemonSets typically provide node-local services that should run on the Node even if it is being drained of workload applications.

Node status

A Node's status contains the following information:

- Addresses
- Conditions
- Capacity and Allocatable
- Info

You can use `kubectl` to view a Node's status and other details:

Search this site

Documentation
Getting started
Concepts
Overview
Cluster Architecture
Nodes
Communication between Nodes and the Control Plane
Controllers
Leases
Cloud Controller Manager
About cgroup v2
Kubernetes Self-Healing
Container Runtime Interface (CRI)
Garbage Collection
Mixed Version Proxy
Containers
Workloads
Services, Load Balancing, and Networking

URL
https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp
Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

- Addresses
- Conditions
- Capacity and Allocatable
- Info

You can use `kubectl` to view a Node's status and other details:

```
kubectl describe node <insert-node-name-here>
```

See Node Status for more details.

Node heartbeats

Heartbeats, sent by Kubernetes nodes, help your cluster determine the availability of each node, and to take action when failures are detected.

For nodes there are two forms of heartbeats:

- Updates to the `.status` of a Node.
- Lease objects within the `kube-node-lease` namespace. Each Node has an associated Lease object.

Node controller

The node controller is a Kubernetes control plane component that manages various aspects of nodes.

The node controller has multiple roles in a node's life. The first is assigning a CIDR block to the node when it is registered (if CIDR assignment is turned on).

The second is keeping the node controller's internal list of nodes up to date with the cloud provider's list of available machines. When running in a cloud environment and whenever a node is unhealthy, the node controller asks the cloud provider if the VM for that node is still available. If not, the node controller deletes the node from its list of nodes.

The third is monitoring the nodes' health. The node controller is responsible for:

URL

https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp

Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

cloud provider's list of available machines. When running in a cloud environment and whenever a node is unhealthy, the node controller asks the cloud provider if the VM for that node is still available. If not, the node controller deletes the node from its list of nodes.

The third is monitoring the nodes' health. The node controller is responsible for:

- In the case that a node becomes unreachable, updating the `Ready` condition in the Node's `.status` field. In this case the node controller sets the `Ready` condition to `Unknown`.
- If a node remains unreachable: triggering API-initiated eviction for all of the Pods on the unreachable node. By default, the node controller waits 5 minutes between marking the node as `Unknown` and submitting the first eviction request.

By default, the node controller checks the state of each node every 5 seconds. This period can be configured using the `--node-monitor-period` flag on the `kube-controller-manager` component.

Rate limits on eviction

In most cases, the node controller limits the eviction rate to `--node-eviction-rate` (default 0.1) per second, meaning it won't evict pods from more than 1 node per 10 seconds.

The node eviction behavior changes when a node in a given availability zone becomes unhealthy. The node controller checks what percentage of nodes in the zone are unhealthy (the `Ready` condition is `Unknown` or `False`) at the same time:

- If the fraction of unhealthy nodes is at least `--unhealthy-zone-threshold` (default 0.55), then the eviction rate is reduced.
- If the cluster is small (i.e. has less than or equal to `--large-cluster-size-threshold` nodes - default 50), then evictions are stopped.
- Otherwise, the eviction rate is reduced to `--secondary-node-eviction-rate` (default 0.01) per second.

The reason these policies are implemented per availability zone is because one availability zone might become partitioned from the control plane while the others remain connected. If your cluster does not span multiple cloud provider availability zones, then the eviction mechanism does not take per-zone unavailability into account.

A key reason for spreading your nodes across availability zones is so that the workload can be shifted to healthy zones when one entire zone goes down. Therefore, if all

URL

https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp

Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

The reason these policies are implemented per availability zone is because one availability zone might become partitioned from the control plane while the others remain connected. If your cluster does not span multiple cloud provider availability zones, then the eviction mechanism does not take per-zone unavailability into account.

A key reason for spreading your nodes across availability zones is so that the workload can be shifted to healthy zones when one entire zone goes down. Therefore, if all nodes in a zone are unhealthy, then the node controller evicts at the normal rate of `--node-eviction-rate` . The corner case is when all zones are completely unhealthy (none of the nodes in the cluster are healthy). In such a case, the node controller assumes that there is some problem with connectivity between the control plane and the nodes, and doesn't perform any evictions. (If there has been an outage and some nodes reappear, the node controller does evict pods from the remaining nodes that are unhealthy or unreachable).

The node controller is also responsible for evicting pods running on nodes with `NoExecute` taints, unless those pods tolerate that taint. The node controller also adds taints corresponding to node problems like node unreachable or not ready. This means that the scheduler won't place Pods onto unhealthy nodes.

Resource capacity tracking

Node objects track information about the Node's resource capacity: for example, the amount of memory available and the number of CPUs. Nodes that self register report their capacity during registration. If you manually add a Node, then you need to set the node's capacity information when you add it.

The Kubernetes scheduler ensures that there are enough resources for all the Pods on a Node. The scheduler checks that the sum of the requests of containers on the node is no greater than the node's capacity. That sum of requests includes all containers managed by the kubelet, but excludes any containers started directly by the container runtime, and also excludes any processes running outside of the kubelet's control.

> **Note:**
> If you want to explicitly reserve resources for non-Pod processes, see reserve resources for system daemons.

Node topology

URL
https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp
Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

> **Note:**
> If you want to explicitly reserve resources for non-Pod processes, see reserve resources for system daemons.

Node topology

FEATURE STATE: Kubernetes v1.27 [stable] (enabled by default: true)

If you have enabled the `TopologyManager` feature gate, then the kubelet can use topology hints when making resource assignment decisions. See Control Topology Management Policies on a Node for more information.

Swap memory management

FEATURE STATE: Kubernetes v1.30 [beta] (enabled by default: true)

To enable swap on a node, the `NodeSwap` feature gate must be enabled on the kubelet (default is true), and the `--fail-swap-on` command line flag or `failSwapOn` configuration setting must be set to false. To allow Pods to utilize swap, `swapBehavior` should not be set to `NoSwap` (which is the default behavior) in the kubelet config.

> **Warning:**
> When the memory swap feature is turned on, Kubernetes data such as the content of Secret objects that were written to tmpfs now could be swapped to disk.

A user can also optionally configure `memorySwap.swapBehavior` in order to specify how a node will use swap memory. For example,

```
memorySwap:
  swapBehavior: LimitedSwap
```


URL
https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp
Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

A user can also optionally configure `memorySwap.swapBehavior` in order to specify how a node will use swap memory. For example,

```
memorySwap:
  swapBehavior: LimitedSwap
```

- `NoSwap` (default): Kubernetes workloads will not use swap.
- `LimitedSwap` : The utilization of swap memory by Kubernetes workloads is subject to limitations. Only Pods of Burstable QoS are permitted to employ swap.

If configuration for `memorySwap` is not specified and the feature gate is enabled, by default the kubelet will apply the same behaviour as the `NoSwap` setting.

With `LimitedSwap` , Pods that do not fall under the Burstable QoS classification (i.e. `BestEffort` / `Guaranteed` Qos Pods) are prohibited from utilizing swap memory. To maintain the aforementioned security and node health guarantees, these Pods are not permitted to use swap memory when `LimitedSwap` is in effect.

Prior to detailing the calculation of the swap limit, it is necessary to define the following terms:

- `nodeTotalMemory` : The total amount of physical memory available on the node.
- `totalPodsSwapAvailable` : The total amount of swap memory on the node that is available for use by Pods (some swap memory may be reserved for system use).
- `containerMemoryRequest` : The container's memory request.

Swap limitation is configured as: `(containerMemoryRequest / nodeTotalMemory) * totalPodsSwapAvailable` .

It is important to note that, for containers within Burstable QoS Pods, it is possible to opt-out of swap usage by specifying memory requests that are equal to memory limits. Containers configured in this manner will not have access to swap memory.

Swap is supported only with **cgroup v2**, cgroup v1 is not supported.

For more information, and to assist with testing and provide feedback, please see the blog-post about Kubernetes 1.28: NodeSwap graduates to Beta1, KEP-2400 and its design proposal.

Search this site

▸ Documentation
▸ Getting started
▾ Concepts
▸ Overview
▾ Cluster Architecture
Nodes
Communication between Nodes and the Control Plane
Controllers
Leases
Cloud Controller Manager
About cgroup v2
Kubernetes Self-Healing
Container Runtime Interface (CRI)
Garbage Collection
Mixed Version Proxy
▸ Containers
▸ Workloads
▸ Services, Load Balancing, and Networking

URL

https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp

Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

Swap is supported only with **cgroup v2**, cgroup v1 is not supported.

For more information, and to assist with testing and provide feedback, please see the blog-post about Kubernetes 1.28: NodeSwap graduates to Beta1, KEP-2400 and its design proposal.

What's next

Learn more about the following:

- Components that make up a node.
- API definition for Node.
- Node section of the architecture design document.
- Graceful/non-graceful node shutdown.
- Node autoscaling to manage the number and size of nodes in your cluster.
- Taints and Tolerations.
- Node Resource Managers.
- Resource Management for Windows nodes.

Feedback

Was this page helpful?

Yes No

Last modified April 26, 2024 at 3:39 PM PST: cluster-autoscaling -> node-autoscaling clean-up (dc530ffd6a)

URL
https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp
Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)

URL

https://kubernetes.io/docs/concepts/architecture/nodes/

Timestamp

Fri May 23 2025 18:54:05 GMT-0500 (Central Daylight Time)